Objective: You're going to implement a data mining algorithm that is commonly used in the frequent pattern mining domain. Frequent item sets **must** be created using the map method. Also, there is a bonus question if you'd like some extra points.

Theoretical Background: In data mining, there's a domain called frequent pattern mining which finds the items that coexists under a certain objective function. In this assignment the objective function will be a receipt list (sales slip). One of the most used algorithms in frequent pattern mining is the Apriori algorithm [1]. Apriori algorithm consists of two recursive steps. First, algorithm calculates the subsets and then it calculates the frequencies of those subsets. Normally the creation of those subsets done in a recursive manner but you're going to achieve it with map function.

Example: Let's assume we have the following items in our item list. There is a total of 4 unique items in this list.

[A, B, C, D]

First, the algorithm will calculate n-element subsets. 4 items mean there will be 4 categories of subsets. 1-element, 2-element, 3-element, 4-element.

Generation of 1 element subsets:

1 element subsets are easy to generate since they can be generated from individual unique items. 1-element subsets will be:

A

B

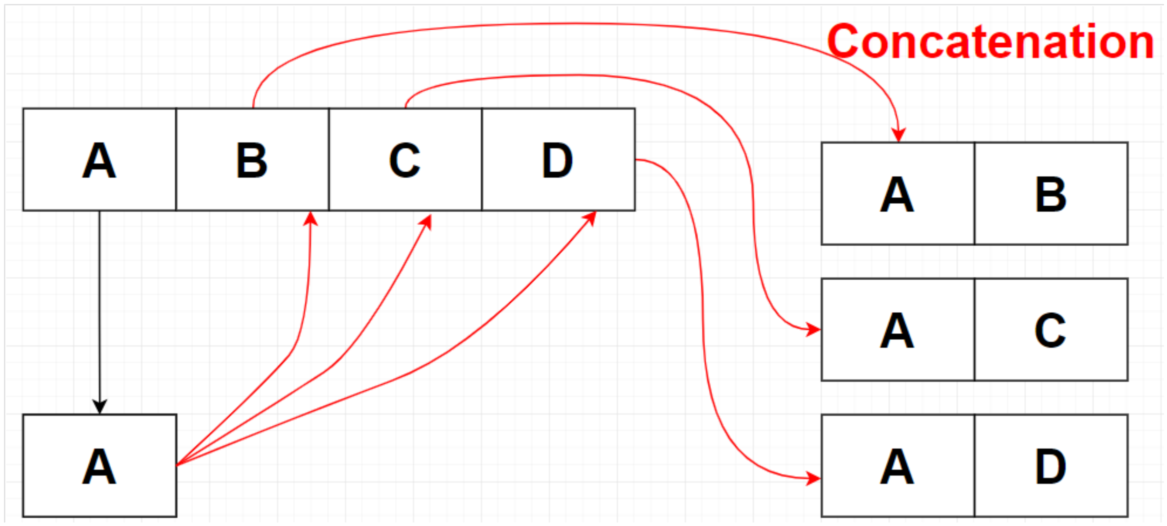
C

D

Generation of 2 element subsets:

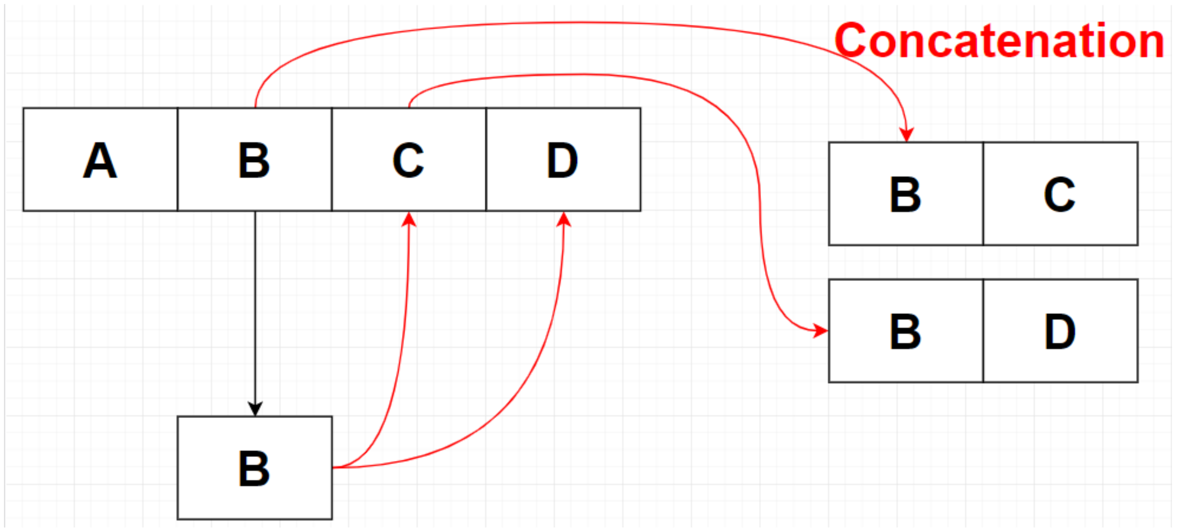
Each subsequent subset is generated from the previous subset. 2-element subsets will be generated using the 1-element subsets, 3-element subsets will be generated using the 2-element subsets etc. 2-element subsets can be obtained in 3 steps. You can see the process in the figures given below.

1)



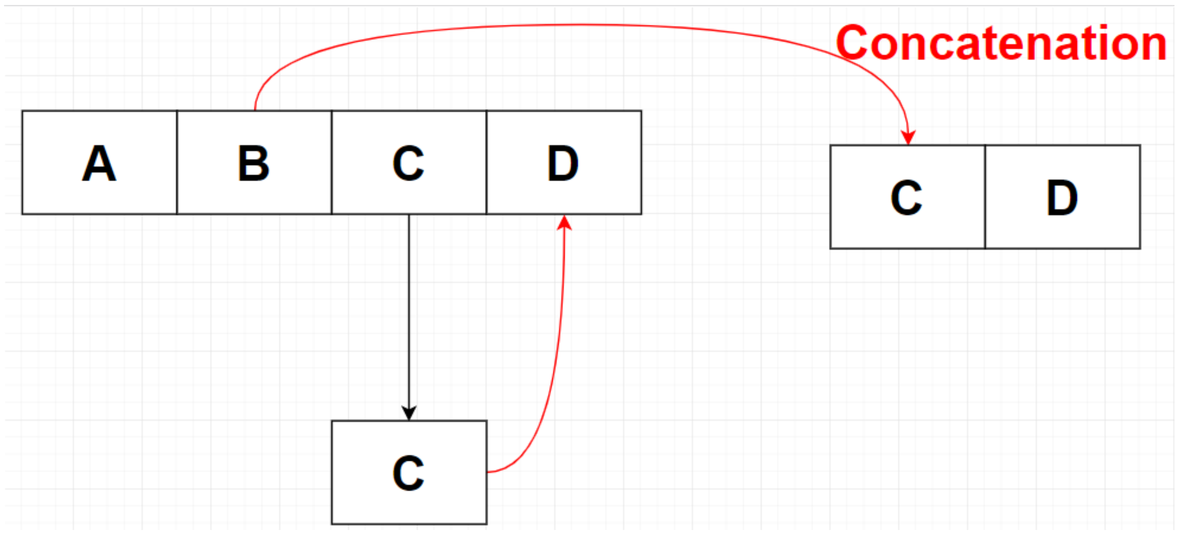
After **mapping** the item “A” with the other items, we get the “AB”, “AC” and “AD”.

2)



After mapping (concatenating) the item "B" with the other items, we get "BC" and "BD"

3)



After mapping (concatenating) the item C with the other items, we get CD. So, after these steps, 2-element subset is created and it consists of the following items:

AB

AC

AD

BC

BD

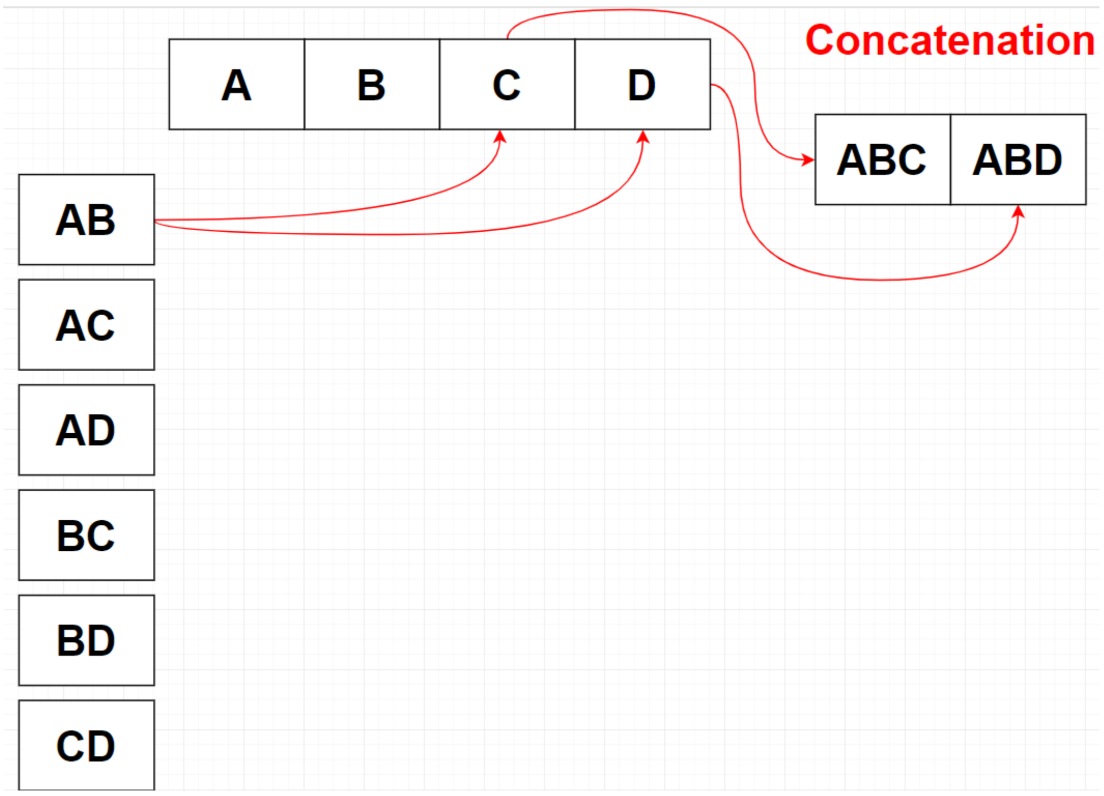
CD

As you can see when we move to the next item, we don't take its previous combinations. Ex: We've generated "AB", when we move to the "B", we don't take "A" because "AB" (or "BA" doesn't matter) is already present in our set. So, when we're at the second item, we are mapping from the third item, when we're at the third item, we are mapping from the fourth item etc.

Generation of 3 element subsets:

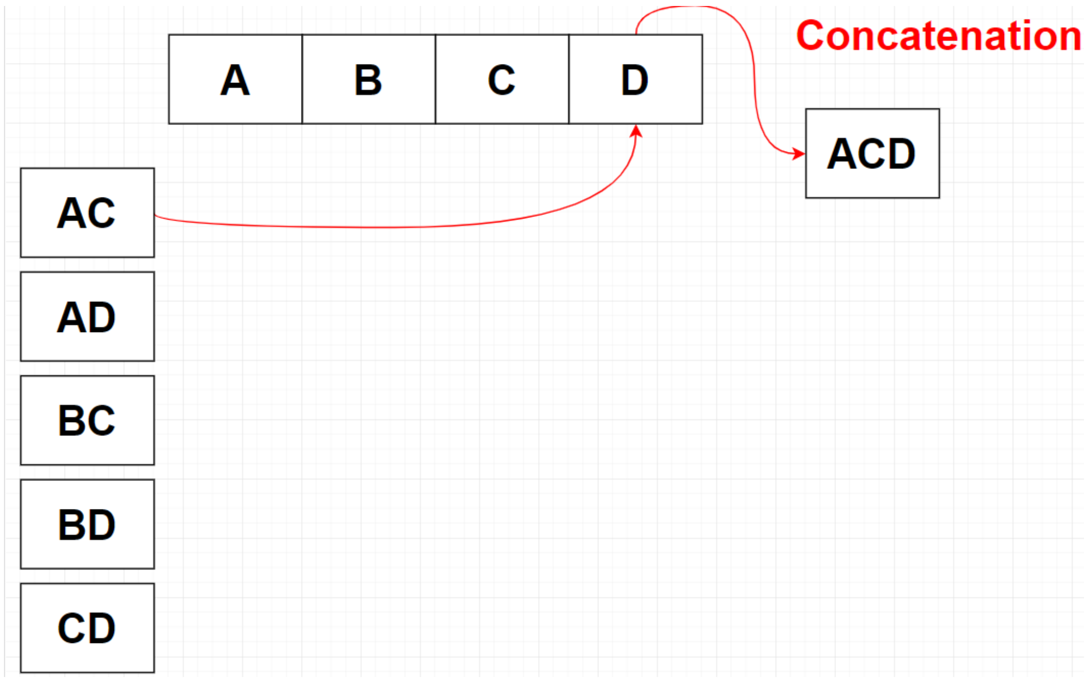
3-element subsets are generated from the 2-element subsets. We have 6 elements in the 2-element subset. Pay attention to the last element when generating the 3-element or higher subsets. Generation of the 3-element subset can be seen in the figure given below.

1)



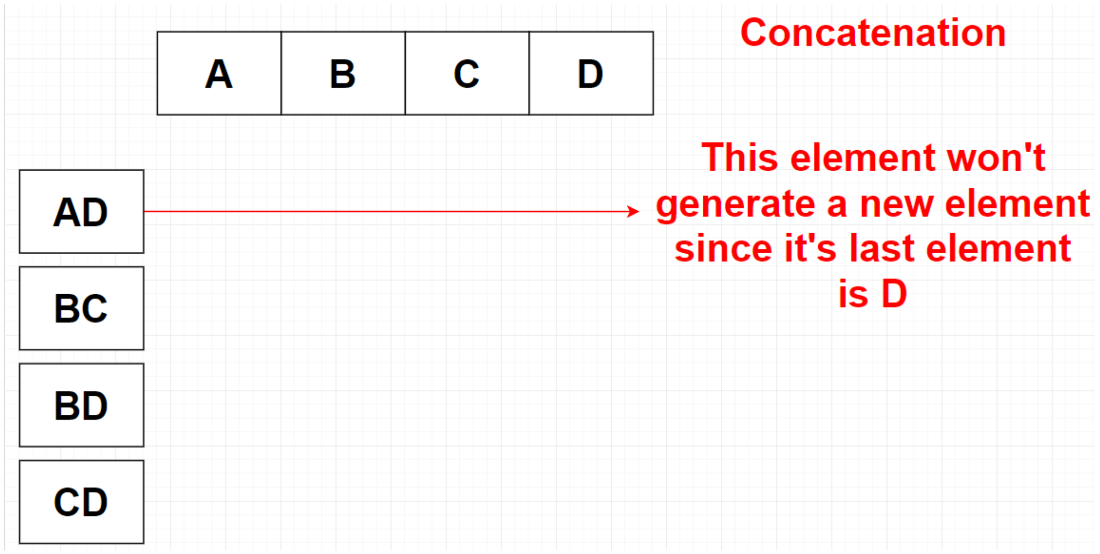
After this step we get, "ABC" and "ABD". Be careful about the last item in the 2-element subset. When creating items from the element "AB", "B" is the last element. So, we'll use the elements next to "B" to create subset elements (subsequent elements are "C" and "D)".

2)



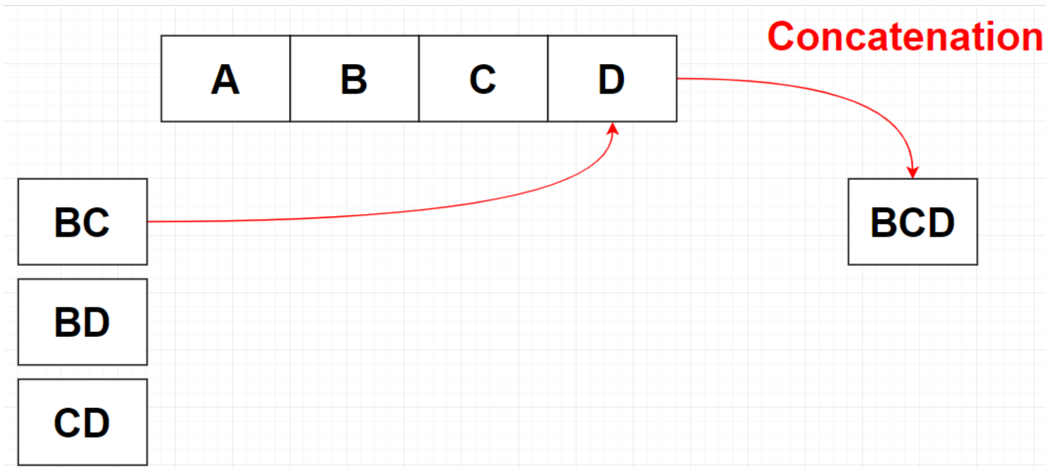
After this step we get ACD.

3)


A
B
C
D
Concatenation
This element won't generate a new element since it's last element is D
AD
BC
BD
CD

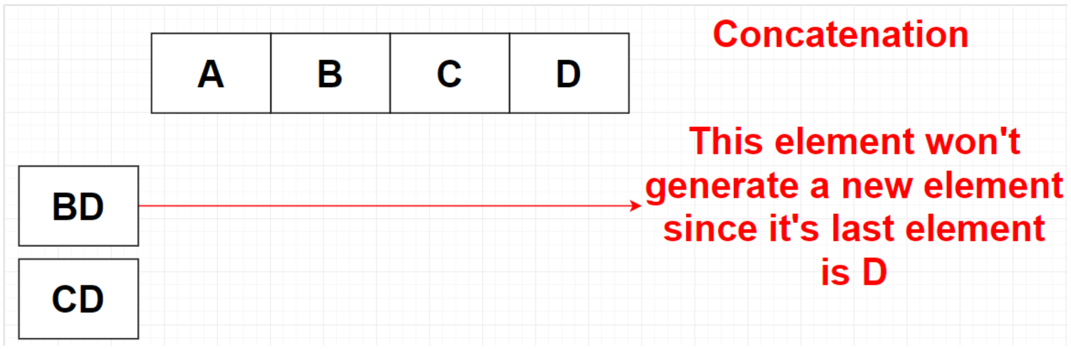
Pay attention “AD”s last element is “D” which is the last element of the unique element set (item_list). So, this won’t generate a new item.

4)



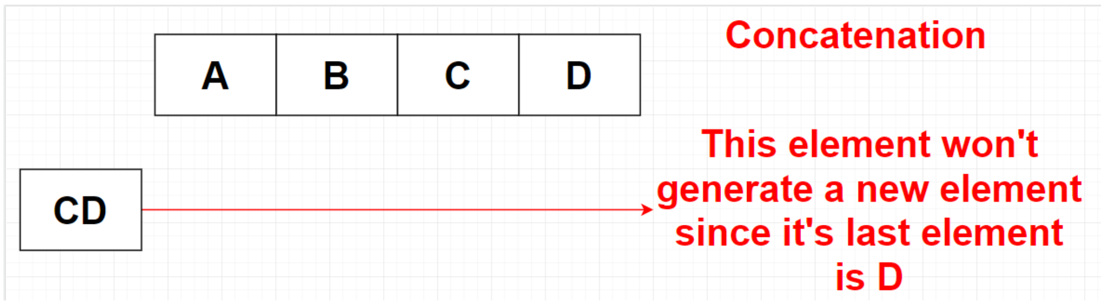
After this step we get, “BCD”.

5)


Concatenation
A
B
C
D
BD
CD
This element won't generate a new element since it's last element is D

“BD” won’t create a 3-element item because it ends with the last item “D”.

6)


Concatenation
A
B
C
D
CD
This element won't generate a new element since it's last element is D

“CD” won’t create a 3-element item because it ends with the last item ‘D’. So, after these steps 3-element subset is created and it consists of the following items:

ABC

ABD

ACD

BCD

Generation of 4 element subsets:

There'll be only one 4-element subset. Because only the element "ABC" is capable of creating a new element and the item list has only 4 elements (A, B, C and D). So, the only item 4-element subset has:

ABCD

All of the n-element subsets for the provided example can be seen in the following table:

1-element subset	2-element subset	3-element subset	4-element subset
A	AB	ABC	ABCD
B	AC	ABD	
C	AD	ACD	
D	BC	BCD	
	BD		
	CD		

Question: In this homework, you're going to implement an algorithm that finds the frequent item subsets of a given list. For this assignment the provided list is given below (You can use integers in your items list to make your job easier):

```
item_list = [("kola"),("fıstık"),("su"),("ayran"),("gazoz"),("fanta")]
```

As you can see there are 6 unique items in the given list. That means there will be 1-element, 2-element, 3-element, 4-element, 5-element and 6-element subsets. You're going to write a **GENERIC** function that utilizes the **map** function in order to create n-element subsets. Remember, you **need to use** the map function, otherwise your homework will not be graded. I expect you to write a function signature similar to this:

```
def create_frequency_sets(item_list, previous_frequency_set):

    """This function takes the distinct item list and previous frequency set and generates
    a frequency set. Generated frequency set consist of (n+1) elements if you accept previous
    frequency set elements as n"""
```

As you can see the function takes the item_list and the previous_frequency_set and generates the current n-element subset. Ex: It takes item_list and item_list (Please pay attention, for the first iteration there are no subsets. So, at first it takes item_list two times to generate 2-element subset. You may implement this part differently) and generates 2-element subset, then it takes item_list and 2-element subset and generates 3-element subset, then it takes item_list and 3-element subset and generates 4-element subset and so on. I expect you to construct a code block similar to the following:

```
#Creating the frequency sets.
twos = create_frequency_sets(item_list, item_list)

threes = create_frequency_sets(item_list, twos)

fours = create_frequency_sets(item_list, threes)

fives = create_frequency_sets(item_list, fours)

six = create_frequency_sets(item_list, fives)
```

As you can see same function is used to create all the subsets. Subsets are created using the previous version of themselves. Remember, your function should be generic, you can use loops inside of your function but it **MUST** utilize the map functionality. All of the subsets are provided below for the question so you can check your algorithm's outputs:

1-element subset:

```
('kola',)
('fıstık',)
('su',)
('ayran',)
('gazoz',)
('fanta',)
```

2-element subset:

```
('kola', 'fıstık')
('kola', 'su')
('kola', 'ayran')
('kola', 'gazoz')
('kola', 'fanta')
('fıstık', 'su')
('fıstık', 'ayran')
('fıstık', 'gazoz')
('fıstık', 'fanta')
('su', 'ayran')
('su', 'gazoz')
('su', 'fanta')
('ayran', 'gazoz')
('ayran', 'fanta')
('gazoz', 'fanta')
```

3-element subset:

```
('kola', 'fıstık', 'su')
('kola', 'fıstık', 'ayran')
('kola', 'fıstık', 'gazoz')
('kola', 'fıstık', 'fanta')
('kola', 'su', 'ayran')
('kola', 'su', 'gazoz')
('kola', 'su', 'fanta')
('kola', 'ayran', 'gazoz')
('kola', 'ayran', 'fanta')
('kola', 'gazoz', 'fanta')
('fıstık', 'su', 'ayran')
('fıstık', 'su', 'gazoz')
('fıstık', 'su', 'fanta')
('fıstık', 'ayran', 'gazoz')
('fıstık', 'ayran', 'fanta')
('fıstık', 'gazoz', 'fanta')
('su', 'ayran', 'gazoz')
('su', 'ayran', 'fanta')
('su', 'gazoz', 'fanta')
('ayran', 'gazoz', 'fanta')
```

4-element subset:

```
('kola', 'fıstık', 'su', 'ayran')
('kola', 'fıstık', 'su', 'gazoz')
('kola', 'fıstık', 'su', 'fanta')
('kola', 'fıstık', 'ayran', 'gazoz')
('kola', 'fıstık', 'ayran', 'fanta')
('kola', 'fıstık', 'gazoz', 'fanta')
('kola', 'su', 'ayran', 'gazoz')
('kola', 'su', 'ayran', 'fanta')
('kola', 'su', 'gazoz', 'fanta')
('kola', 'ayran', 'gazoz', 'fanta')
('fıstık', 'su', 'ayran', 'gazoz')
('fıstık', 'su', 'ayran', 'fanta')
('fıstık', 'su', 'gazoz', 'fanta')
('fıstık', 'ayran', 'gazoz', 'fanta')
('su', 'ayran', 'gazoz', 'fanta')
```

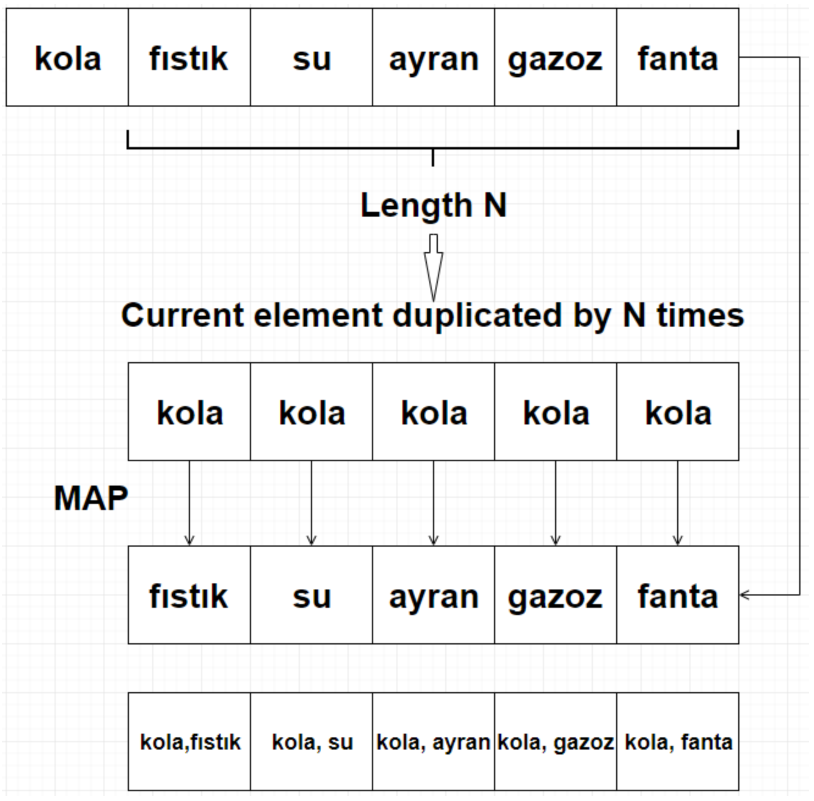
5-element subset:

```
('kola', 'fıstık', 'su', 'ayran', 'gazoz')
('kola', 'fıstık', 'su', 'ayran', 'fanta')
('kola', 'fıstık', 'su', 'gazoz', 'fanta')
('kola', 'fıstık', 'ayran', 'gazoz', 'fanta')
('kola', 'su', 'ayran', 'gazoz', 'fanta')
('fıstık', 'su', 'ayran', 'gazoz', 'fanta')
```

6-element subset:

```
('kola', 'fıstık', 'su', 'ayran', 'gazoz', 'fanta')
```

Hint About the Mapping: When you're generating a subset, you can take a portion from the item list. This portion starts from the subset's last item. Calculate this portion's size and name it as n. Generate a list full of the current subset item that has a length of n. Then map those two lists into a new list. You may want to use list/tuple unpacking. So, have a look at that. The following figure explains the mapping concept:

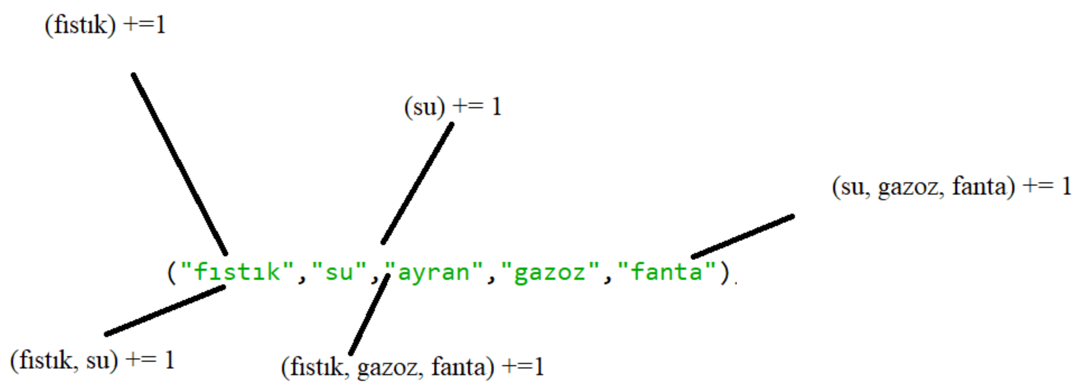


Bonus Question: Apriori algorithm is far from complete, in the bonus question you can complete it to the fullest. Second step of the apriori algorithm consists of counting each elements occurance rate. You've already calculated each possible subset for every element in the first question. You're going to count each subset's occurrences. You're going to write a function to count the occurrences. This function will be sent to a **reduce** function as a parameter. In order to get a bonus, you **must** use the **reduce** function. I've calculated each subset's frequency by adding them into a dictionary, you can use the same logic. There will be a receipt list provided to you, you'll calculate the subset occurrence rates on this list. This list will be available in a pickle format [2]. Load the pickled list and use it as your algorithm's validator. The receipt list has 30 items in it and it already sorted, prepared for the apriori algorithm. You only need to do the counting process with the reduce. The receipt list looks like this:

```
receipt_list = [ ("kola", "su", "fanta"), ("kola", "fıstık"), ("kola","su","gazoz","fanta"),("kola","fanta"),
                ("kola","gazoz"),("kola","su"),("kola","fıstık","su"),("kola","fıstık"),
                ("fıstık","su","gazoz"),("ayran","gazoz"),("fıstık","gazoz"),("fıstık","ayran", "gazoz"),
                ("su"),("su","fanta"),("su","gazoz"),("gazoz"),("gazoz","fanta"),("fıstık","ayran","fanta"),
                ("kola","ayran","fanta"),("fanta"),("su","fanta"),("gazoz"),("su","gazoz"),("fıstık","su","gazoz"),
                ("kola","su"),("fıstık","su","ayran","gazoz","fanta"),("kola","su","ayran","fanta"),
                ("fıstık","ayran","gazoz","fanta"),("kola","su","ayran","gazoz","fanta"),
                ("kola","fıstık","su","ayran","gazoz","fanta")]
```

You need to count each element's occurrence rates from that receipt list. But be careful, one entry may belong to a lot of subset elements.

Example:


(fıstık) +=1
(su) += 1
(su, gazoz, fanta) += 1
("fıstık","su","ayran","gazoz","fanta").
(fıstık, su) += 1
(fıstık, gazoz, fanta) +=1

As you can see an entry may contain more than one element. So, you need to increment the occurrence of every possible element that may reside in a single entry. When you run your reducer on the provided example you should find the following occurrence rates:

```
('kola',) : 13
('fıstık',) : 11
('su',) : 16
('ayran',) : 9
('gazoz',) : 16
('fanta',) : 14
('kola', 'fıstık') : 4
('kola', 'su') : 8
('kola', 'ayran') : 4
('kola', 'gazoz') : 4
('kola', 'fanta') : 7
('fıstık', 'su') : 5
('fıstık', 'ayran') : 5
('fıstık', 'gazoz') : 7
('fıstık', 'fanta') : 4
('su', 'ayran') : 4
('su', 'gazoz') : 8
('su', 'fanta') : 8
('ayran', 'gazoz') : 6
('ayran', 'fanta') : 7
('gazoz', 'fanta') : 6
('kola', 'fıstık', 'su') : 2
('kola', 'fıstık', 'ayran') : 1
('kola', 'fıstık', 'gazoz') : 1
('kola', 'fıstık', 'fanta') : 1
('kola', 'su', 'ayran') : 3
('kola', 'su', 'gazoz') : 3
('kola', 'su', 'fanta') : 5
('kola', 'ayran', 'gazoz') : 2
('kola', 'ayran', 'fanta') : 4
('kola', 'gazoz', 'fanta') : 3
('fıstık', 'su', 'ayran') : 2
('fıstık', 'su', 'gazoz') : 4
('fıstık', 'su', 'fanta') : 2
('fıstık', 'ayran', 'gazoz') : 4
('fıstık', 'ayran', 'fanta') : 4
```

```
('fıstık', 'gazoz', 'fanta') : 3
('su', 'ayran', 'gazoz') : 3
('su', 'ayran', 'fanta') : 4
('su', 'gazoz', 'fanta') : 4
('ayran', 'gazoz', 'fanta') : 4
('kola', 'fıstık', 'su', 'ayran') : 1
('kola', 'fıstık', 'su', 'gazoz') : 1
('kola', 'fıstık', 'su', 'fanta') : 1
('kola', 'fıstık', 'ayran', 'gazoz') : 1
('kola', 'fıstık', 'ayran', 'fanta') : 1
('kola', 'fıstık', 'gazoz', 'fanta') : 1
('kola', 'su', 'ayran', 'gazoz') : 2
('kola', 'su', 'ayran', 'fanta') : 3
('kola', 'su', 'gazoz', 'fanta') : 3
('kola', 'ayran', 'gazoz', 'fanta') : 2
('fıstık', 'su', 'ayran', 'gazoz') : 2
('fıstık', 'su', 'ayran', 'fanta') : 2
('fıstık', 'su', 'gazoz', 'fanta') : 2
('fıstık', 'ayran', 'gazoz', 'fanta') : 3
('su', 'ayran', 'gazoz', 'fanta') : 3
('kola', 'fıstık', 'su', 'ayran', 'gazoz') : 1
('kola', 'fıstık', 'su', 'ayran', 'fanta') : 1
('kola', 'fıstık', 'su', 'gazoz', 'fanta') : 1
('kola', 'fıstık', 'ayran', 'gazoz', 'fanta') : 1
('kola', 'su', 'ayran', 'gazoz', 'fanta') : 2
('fıstık', 'su', 'ayran', 'gazoz', 'fanta') : 2
('kola', 'fıstık', 'su', 'ayran', 'gazoz', 'fanta') : 1
```

Good Luck.

Important Points:

- Similarity check will be applied to your scripts. Don’t cheat, you’ll get a zero if you do.
- Write a generic code. Your code must be working on different receipt lists.
- Don’t miss the deadline.

References:

[1]:https://www.wikizeroo.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvQXByaW9yaV9hbGdvcml0aG0

[2]: https://docs.python.org/3/library/pickle.html